A Unification of Artificial Potential Function Guidance and Optimal Trajectory Planning

Carl Glen Henshaw
Naval Center for Space Technology
U.S. Naval Research Laboratory
ghenshaw@space.nrl.navy.mil
(202) 767-1196

January 10, 2005

Artificial potential function guidance (APFG) has been widely and successfully used to solve motion planning problems across a variety of domains. APFG is popular, in large part, due to its simplicity and speed. However, APFG is known to have several limitations. Notably, it may not find a free trajectory even where one exists, and in most cases APFG trajectories are highly suboptimal. In contrast, fueled by the development of new numerical optimization algorithms, there has been considerable recent interest in the use of a variety of optimal trajectory planners which do not exhibit these limitations. These algorithms hinge on finding a trajectory which minimizes a given cost functional. When successful these algorithms find very high–quality trajectories, but when applied to nonlinear systems and cluttered environments, they often run too slowly for use in real–time systems, and providing rigorous convergence guarantees for such algorithms may be difficult. This paper demonstrates an apparently unappreciated relationship between APFG and a form of optimal planning called receding horizon planning. This understanding may be used to derive trajectory planning algorithms which fit into an anytime planning framework, which may allow high–quality trajectories to be calculated with guaranteed real–time performance. Preliminary results for a receding horizon planner applied to planar holonomic robotic navigation problem and to a realistic orbital robotic arm grapple maneuver are presented.

1 Introduction

A reliable, high quality, real–time motion planning algorithm is one of the holy grails of robotics research. Motion planning is necessary for robots to find their way around their environments while limiting the amount of time or fuel required to perform their required tasks. In the case of robots which operate in cluttered environments, motion planning is necessary to allow the robot to navigate while avoiding collisions with obstacles or other robots. In some cases, such as with behavioristic systems, the division between trajectory planning, lower level control algorithms, and higher level goal seeking algorithms may be quite blurred. With most successfully fielded systems, however, the trajectory planner constitutes a distinct algorithm, and much effort has gone into developing such algorithms which are fast, complete, correct, and which find time– or length–optimal trajectories. An additional problem has been to ensure that the resulting trajectories are feasible, that is, that they satisfy any dynamic and kinematic constraints for the system in question.

Artificial Potential Function Guidance (APFG) is among the most popular robotic trajectory planners. APFG planners are based on the design of a potential function which has a global minimum centered at the goal and local maxima over obstacles [6], [7], [17]. The trajectory is then generated by performing gradient descent on the potential function. APFG planners are simple to design, execute very quickly, and are applicable to a wide variety of problem domains. However, APFG planners have several known limitations. Chief among these is that it is difficult to design a potential function that is free of local minima, which can cause the gradient descent step to terminate before reaching the goal [11, p. 296]. Another limitation is that there appears to be no straightforward way of including optimization criteria in such a planner; as a consequence the resulting trajectories may be highly suboptimal in terms of fuel use, path length, or other quantities of interest. Finally, enforcing dynamic and/or kinematic constraints using potential functions may be problematic.

Recently, optimal trajectory planning has seen increasing use in robotics [21], [16], [23], [10], [14]. Optimal trajectory planning entails the design of a cost functional, a mathematical expression that “grades” trajectories on the basis of fuel use, time, path length, or any other mathematically quantifiable term of interest to the designer. One of a variety of optimization techniques is then applied to determine the trajectory which minimizes the cost functional; the resulting trajectory is optimal in terms of the specified quantities of interest. In addition, it is easy to incorporate dynamic constraints, thruster constraints, and very complex boundary value constraints into an optimal trajectory planner. Unfortunately, though, optimal trajectory planners are often computationally intensive, especially when used for obstacle avoidance problems for a system with nonlinear kinematics and/or dynamics, which necessitates the use of a nonlinear numerical optimization algorithm [5]. As a consequence their utility as real–time trajectory planners may be limited.

In order to overcome the computational challenges associated with optimal trajectory planning, the use of receding horizon control has recently been suggested [1]. A receding horizon planner is essentially an optimal planner except that the cost functional is formulated over the fixed time period $(t, t+h)$ instead of (t_0, t_f), where it is assumed that h is much smaller than $t_f - t_0$. A step size $\delta < h$ is chosen, and the trajectory from t to $t+\delta$ is traversed; simultaneously, the planner "replans" by generating a trajectory from $t+\delta$ to $t+h+\delta$. Because fewer computations are usually required to minimize a cost functional as the planning horizon becomes smaller, receding horizon planners generally execute much more quickly than end-to-end optimal trajectory planners while retaining many of their desirable aspects. In addition, receding horizon planners can respond more quickly to unpredicted obstacles or other environmental factors that may not be known with precision at t_0. However, because the convergence qualities of nonlinear minimization algorithms are notoriously difficult to analyze, even a receding horizon planner may not always result in real–time performance.

The primary focus of this paper is the description of a technique which may allow the development of receding horizon planners with real–time performance guarantees for virtually any physical system of interest, including systems with nonlinear dynamics and kinematics. This technique is a straightforward outgrowth of the realization that receding horizon planning becomes identical to APFG as the planning horizon h approaches zero. The paper is organized as follows: Section 2 describes APFG in detail. Section 3 describes both "standard" optimal planning and receding horizon planning. Section 4 demonstrates the relationship between APFG and optimal planning. Section 5 discusses how this relationship may be utilized to fit receding horizon planning into an anytime planning framework, where a minimally acceptable trajectory can be generated within a known short period of time and then improved upon for however much time is available. Section 6 shows the results for a receding horizon planner for a simple planar trajectory planning problem and a more realistic orbital dexterous robotic arm grapple. Finally, Section 7 presents conclusions and topics for future work.

2 Artificial Potential Function Guidance

APFG planners first achieved popularity with the dissertation of Khatib [6], who suggested them as a collision avoidance algorithm for articulated robot arms. In the simplest formulation, an APFG planner is based on a potential function which consists of attractive "bowl–shaped" term centered on the goal position and repulsive "hump–shaped" terms over each of the environmental obstacles:

$$G(\mathbf{x}) = A(\mathbf{x}, \mathbf{x}_d) + \beta \sum_{j=0}^{n} R(\mathit{dist}(\mathbf{x}, c_j)) \tag{1}$$

where $A(\cdot)$ is the attractive term, $R(\cdot)$ is the repulsive term, $\mathbf{x}$ is the system position vector, $\mathbf{x}_d$ is the goal position, and c_j represents the set of points occupied by obstacle j. Typically, $A(\cdot)$ is quadratic,

$$A(\mathbf{x}, \mathbf{x}_d) = \|\mathbf{x} - \mathbf{x}_d\|^2, \tag{2}$$

$\mathit{dist}(\mathbf{x}(t), c_j)$ represents the distance between $\mathbf{x}(t)$ and the boundary of c_j, and $R(\cdot)$ is a radial function such as the inverse of the distance from the robot to the obstacle

$$R(\mathbf{x}, \mathbf{c}_j) = 1/\mathit{dist}(\mathbf{x}(t), c_j)$$

or a Gaussian

$$R(\mathbf{x}, \mathbf{c}_j) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{\frac{-dist(\mathbf{x}(t), c_j)^2}{(2\sigma)^2}}.$$

A sequence of waypoints from $\mathbf{x}$ to $\mathbf{x}_d$ is calculated by performing gradient descent on the potential function:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \left.\frac{\partial G(\mathbf{x})}{\partial \mathbf{x}}\right|_{x_i}$$

where, typically, $\gamma \ll 1$. As $i \rightarrow \infty$, it is expected that $\mathbf{x}_i \rightarrow \mathbf{x}_d$. If β is sufficiently large and γ is sufficiently small, it can be guaranteed that the resulting trajectory will be free of collisions [7]. Also, note that the computational complexity required to calculate the gradient $\frac{\partial G}{\partial \mathbf{x}}$ is only $O(n)$ in the dimensionality of the trajectory space and in the number of obstacles (assuming an $O(n)$ distance calculation routine). Therefore, it is usually possible to determine hard real–time performance guarantees for an APFG planner.

This formulation has several well–known limitations. One major problem with the naive formulation is that $G(\cdot)$ may have local minima in which the gradient descent algorithm can become stuck, leading to a failure of the waypoints $\mathbf{x}_i$ to converge to the goal position $\mathbf{x}_d$. This typically occurs when two or more obstacles are close to each other. Considerable effort has been devoted to finding potential functions which are free of extraneous local minimum. Koditschek et al [17] were the first to generate such a potential function. Several different formulations have since been proposed [22], [8]. Notably, most of these formulations require the potential function to be calculated a priori over the entire domain of interest [2]. This calculation may be quite complex, which somewhat diminishes the simplicity that is the appeal of the naive formulation, and also reduces the calculation speed of the algorithm.

Since APFG trajectories consist of waypoints, the trajectory must be parameterized so that it is feasible. Interpreted in the simplest manner, each successive waypoint is reached after a fixed δt, so that at t_i, $\mathbf{x} = \mathbf{x}_i$. The path taken between each waypoint is often assumed to be linear, so that the entire trajectory is piecewise linear, and the velocity achieved at each waypoint is often defined implicitly by whatever lower level control algorithm is used. However, such a scheme may yield an unfeasible trajectory. For instance, when using a quadratic attractive potential such as Equation 2, in the absence of obstacles the gradient becomes steeper as the distance from the goal position increases. Therefore, if the robot begins at rest (or, indeed, with any velocity vector other than one that happens to point directly at the waypoint), a large initial velocity spike will result which may be higher than the system actuators can accommodate. In addition, many systems of interest, such as aircraft or spacecraft, are unable to follow piecewise linear trajectories. In order to follow a piecewise trajectory most other systems would have to stop completely at each waypoint, which is clearly undesirable from both a fuel use and arrival time standpoint.

To overcome these problems the trajectory must be intelligently parameterized somehow. Parameterization schemes may be as simple as varying the required arrival time and velocity at each waypoint in order to keep the actuation requirements within acceptable limits, or as complex as using the waypoints as the initial trajectory for a trajectory optimization scheme (see below). It may also be necessary to "adjust" the shape of the potential function to ensure that requirements such as minimum turn radii or (for an aircraft) maximum rate of climb are met. Formulations have also been proposed where, instead of waypoints, the APFG planner returns a control input $\mathbf{u}$. These formulations are then guaranteed to satisfy system dynamic constraints; thruster constraints are more problematic, but formulations taking them into account have also been proposed [18], [12], [13]. Unlike the waypoint formulation, however, considerable care must be taken to guarantee that the resulting trajectories are collision free.

A limitation which is less well recognized but just as problematic is that the trajectories computed from APFG planners are generally not optimal in any sense. For a simple system like a wheeled robot this is not usually a problem; but for many types of systems this limitation may eliminate APFG as a viable option. For an orbiting spacecraft, for instance, minimizing fuel use, or at least keeping fuel use low in some sense, is often very important. Similarly, for a robot arm it is often desirable to minimize the joint velocities, and for an aircraft minimizing time of flight or distance traveled may be important. There do not appear to be any extensions to APFG in the literature which attempt to solve this problem.

3 Optimal Trajectory Planning

The basic idea behind optimal trajectory planning is to use one of a variety of numerical methods to find a trajectory which minimizes a cost functional [9], generically formulated as

$$J\left[\mathbf{x}(t), t\right] = F(\mathbf{x}(t_f), \dot{\mathbf{x}}(t_f), t_f) + \int_{t_0}^{t_f} G(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)\, dt \tag{3}$$

There is usually at least one hard constraint, requiring that $\mathbf{x}(t_0) = \mathbf{x}_0$, instituted on admittable solutions. The other endpoint may also be handled as a hard constraint, requiring that $\mathbf{x}(t_f) = \mathbf{x}_f$. Alternatively, $F(\cdot)$ may include a term penalizing the distance from $\mathbf{x}(t_f)$ to $\mathbf{x}_f$ [9, p. 42–45] such as:

$$F(\mathbf{x}(t_f), t_f) = \cdots + \alpha \left\|\mathbf{x}(t_f) - \mathbf{x}_f\right\|^2 + \cdots$$

This is referred to as a soft constraint. Additionally, environmental obstacles may be handled via hard constraints, by requiring that

$$\mathbf{x}(t) \notin \bigcup c_j \ \ \forall\, t \in [t_0, t_f]$$

or via a soft contraint by adding a term to $G(\cdot)$ which penalizes obstacle clearance distance, such as

$$G(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) = \cdots + \beta \sum_{j=1}^{n} R(\mathit{dist}(\mathbf{x}(t), c_j)) + \cdots$$

$G(\cdot)$ may also contain terms which penalize fuel use, path length, or arrival time, or virtually any other mathematically expressible quantity. The optimal trajectory will then "trade off" each of the penalized terms to determine a trajectory which results in the lowest overall score.

The problem of finding the trajectory which minimizes the cost functional subject to the hard constraints may be solved in one of several ways. The classic approach is to augment $G(\cdot)$ using Lagrange multipliers [9, Chapter 5]:

$$G_a(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) = G(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) + \lambda^T(t)\mathbf{f}(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)$$

where the $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)$ term defines the system dynamics and the vector of time–varying Lagrange multipliers $\lambda(t)$ is called the costate. Then it can be shown that the solution to the differential equations

$$\begin{aligned} \dot{\lambda}(t) &= -\frac{\partial G_a(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)}{\partial \mathbf{x}(t)} \\ \dot{\mathbf{x}}(t) &= \frac{\partial G_a(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)}{\partial \lambda(t)} = \mathbf{f}(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) \end{aligned} \tag{4}$$

subject to suitably chosen boundary constraints minimizes the cost functional $J(\cdot)$. This technique is known as the indirect approach. This approach has the advantage that is easy to derive Euler–Lagrange equations for systems with arbitrary dynamics and actuator constraints. It is easy to incorporate thrusters with limited force output or even thrusters with discrete output levels, for instance. Nonholonomic systems may also be treated.

The particular boundary constraints depend on the particular problem being solved [9, pp. 189-198]; in the simplest case, where $F(\cdot) = 0$ and both $\mathbf{x}_0$ and $\mathbf{x}_f$ are considered hard constraints, the boundary conditions are

$$\begin{aligned} \mathbf{x}(t_0) &= \mathbf{x}_0 \\ \mathbf{x}(t_f) &= \mathbf{x}_f. \end{aligned} \tag{5}$$

If $\mathbf{x}_f$ is a soft constraint or there are other terms in $F(\cdot)$, then the boundary conditions become more complex. For instance, if $\mathbf{x}_f$ is a soft constraint, such as $F(\mathbf{x}(t_f), \dot{\mathbf{x}}(t), t) = \alpha \left\|\mathbf{x}(t_f) - \mathbf{x}_f\right\|^2$, then the necessary boundary conditions are

$$\begin{aligned} \mathbf{x}(t_0) &= \mathbf{x}_0 \\ -\lambda(t_f) &= \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}(t_f), \dot{\mathbf{x}}(t), t) = 2\alpha(\mathbf{x}(t_f) - \mathbf{x}_f) \end{aligned} \tag{6}$$

Other hard constraints may be incorporated by adding additional Lagrange multipliers to Equation 4. It is also possible to choose the optimal arrival time t_f. In all cases, however, the boundary conditions are split, that is, the entire state vector $[\mathbf{x}(t), \lambda(t)]^T$ is not known at either t_0 or t_f; instead, parts of the required state vector are known at t_0 and parts at t_f. Consequently, the Euler–Lagrange equations cannot be solved merely by integrating them forward (or backward) in time; instead, fairly complex iterative numerical algorithms must be utilized to locate a solution which satisfies the differential equations and the boundary conditions.

Recently, an alternative approach to solving the optimal motion planning problem has gained popularity. In this approach, one attempts to approximate the trajectory which minimizes $J[\cdot]$ using a spline or another approximation technique. Consequently, instead of searching for an optimal function $\mathbf{x}(t)$ over the space of all possible functions of time, one searches for a coefficient vector $\mathbf{b}$ which minimizes the cost function

$$J_s(\mathbf{b}) = \int_{t_0}^{t_f} G(\mathbf{b}^T\mathbf{d}(t), \mathbf{b}^T\dot{\mathbf{d}}(t), t)\, dt + F(\mathbf{b}^T\mathbf{d}(t_f), \mathbf{b}^T\dot{\mathbf{d}}(t_f), t_f) \tag{7}$$

where $\mathbf{d}(t)$ is a vector of basis functions. Thus, one can apply vector optimization algorithms such as sequential quadratic programming (SQP), Newton–Gauss, or Levenberg–Marquardt to the problem. This technique is known as the direct approach [24]. The primary conceptual difficulty in this approach is that one must find an approximation space whose members all satisfy the system dynamic constraints (in other words, every member of the space must be a feasible trajectory), because unlike the indirect approach, here the system dynamic constraints are not treated explicitly in the formulation of the optimization problem. For rigid body systems, this happens to be fairly straightforward since rigid body systems are a subset of Hamiltonian systems, and all trajectories with piecewise continuous acceleration profiles are feasible for such systems. Since cubic splines and third-order Hermite splines have piecewise continuous second derivatives, all trajectories parameterized by either of these spline families are feasible for rigid body systems. One must also restrict the search to splines which satisfy the actuator constraints, either by finding a spline space whose members all satisfy these constraints as well, by expressing these constraints as hard constraints on the optimization problem and using a solution technique such as SQP which can solve problems with hard constraints, or by penalizing deviations from the actuator constraints via soft constraints in the cost functional.

Although it is difficult to make any definitive statements about the performance of a particular algorithm for solving optimal motion planning problems without defining the specific system and cost functional to which the algorithm is to be applied, in general direct optimization techniques are felt to be more robust and faster than indirect techniques. Indirect techniques are more flexible, however, because they can at least in principle be applied to nearly any system dynamics and actuator constraints. Both approaches potentially suffer from the problems associated with nonlinear optimization routines. Of these, the limitations of interest here are simply that nonlinear optimization techniques are computationally intensive, and their convergence is typically sensitive to initial conditions. As a result, in the face of nonlinearities such as those imposed by the presence of obstacles in the environment it is very challenging to develop optimal trajectory planners that reliably generate results in the time frames required by most robotics problems. Indeed, Henshaw [5] recently proposed an optimal trajectory planning algorithm for solving minimum-fuel orbital translation/rotation docking maneuvers. The algorithm was capable of solving such problems but tended to require minutes to hours of computation time. Also, even though Henshaw emphasized robustness in the design of his algorithm, the approach often required the assistance of a human operator to find initial conditions which allowed the algorithm to converge.

As a result of the computational complexity and lack of robustness of optimal trajectory planners, Bellingham et al [1], [19], [20] suggested the use of receding horizon planning for robotic trajectory planning problems. Receding horizon planning (also known as model predictive control) [4], [15] has been used since the 1970's, especially in the chemical process industry, as a way to improve the speed of optimal control algorithms. Receding horizon planning is essentially optimal planning except that the cost functional 3 is formulated over a limited time horizon $(t, t+h)$ instead of (t_0, t_f) or (t_0, ∞):

$$J\left[\mathbf{x}(t), t\right] = \int_t^{t+h} G(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)\ dt + E(\mathbf{x}(t+h), \dot{\mathbf{x}}(t+h), t+h) \tag{8}$$

where, here, the terminal cost $F(\cdot)$ has been dropped in favor of a cost-to-go estimate $E(\cdot)$ that is usually designed to estimate the true cost-to-go $\int_{t+h}^{t_f} G(\mathbf{x}(t), \dot{\mathbf{x}}(t), t)\ dt + F(\mathbf{x}(t_f), \dot{\mathbf{x}}(t_f), t_f)$. Operationally, the system does not usually traverse the entire optimal path from t to $t+h$; instead, a step size $\delta < h$ is chosen, the trajectory is traversed from t to $t+\delta$, and the algorithm is repeated with $t+\delta$ as the new t_0. Note that as the horizon h approaches $t_f - t_0$, the performance of the receding horizon algorithm approaches that of the previous optimal planner. Receding horizon planners are usually faster than optimal planners because it generally becomes less computationally expensive to solve optimization problems as the time period over which the optimization occurs becomes smaller. In additino, unlike an optimal planner, a receding horizon planner is not required to calculate the entire trajectory before the system can begin to move; instead, only the step $[t, t+h]$ is required. As the trajectory from $[t, t+\delta]$ is being traversed, the planner can calculate future steps. As a result of these two factors, receding horizon planners can allow the system to respond much more quickly than optimal planners.

Like an optimal planner, a receding horizon planner still needs to use an online numerical optimization routine in order to minimize Equation 8. Bellingham uses Mixed Integer–Linear Programming (MILP), for which there are high quality, fast, robust codes available. Unfortunately, however, MILP is limited to use with linear systems, and Bellingham's technique is therefore unfortunately inapplicable to many systems of interest, including robotic arms and spacecraft attitude problems. To solve trajectory planning problems for nonlinear systems, a nonlinear optimization routine such as Newton–Gauss, Sequential Quadratic Programming, or Levenberg–Marquardt must be used. As mentioned previously, providing a rigorous convergence analysis for nonlinear optimization routines is very difficult. In fact, it is generally not possible to even guarantee that the algorithm will converge at all from a given set of initial conditions. This is clearly an undesirable characteristic for a real–time trajectory planner. MILP has a similar problem, in that although it is more robust than nonlinear opimizers, like nonlinear optimization algorithms the time required to find a solution cannot be guaranteed.

4 Relationship between APFG and Optimal Motion Planning

At first glance, APFG and optimal motion planning appear to be disparate algorithms. APFG is strictly local: it takes into account only the gradient of the potential field at a single point, does no searching, and does not appear to be optimizing anything, either explicitly or implicitly. Optimal motion planning algorithms, on the other hand, are global and perform explicit minimization of a cost functional. However, there is a framework whereby APFG and direct optimal motion planning algorithms can be interpreted as merely the two extremes of receding horizon planning. It is widely recognized that optimal planning and receding horizon planning are closely related, but the relationship between receding horizon planning and APFG is apparently less widely appreciated.

To establish this framework, first note that without loss of generality that a cost functional $J[\cdot]$ may be written as

$$\begin{aligned} J[\mathbf{x}(t),\dot{\mathbf{x}}(t),t] &= F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) + \int_{t_0}^{t_f} G(\mathbf{x}(t),\dot{\mathbf{x}}(t),t)\,dt \\ &= F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) + \int_{t_0}^{t_0+h} G(\mathbf{x}(t),\dot{\mathbf{x}}(t),t)\,dt + \int_{t_0+h}^{t_f} G(\mathbf{x}(t),\dot{\mathbf{x}}(t),t)\,dt \end{aligned} \tag{9}$$

Now assuming that h is sufficiently small, $h = \delta t$, then the integrand of the first integral term can be replaced by its truncated Taylor expansion:

$$\begin{aligned} J[\mathbf{x}(t),\dot{\mathbf{x}}(t),t] &= F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) \\ &\quad + \int_{t_0}^{t_0+\delta t} \left(G(\mathbf{x}(t_0),\dot{\mathbf{x}}(t_0),t_0) + \left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} (\mathbf{x}(t)-\mathbf{x}(t_0)) + \left.\frac{\partial G}{\partial \dot{\mathbf{x}}}\right|_{x_0} (\dot{\mathbf{x}}(t)-\dot{\mathbf{x}}(t_0)) + \left.\frac{\partial G}{\partial t}\right|_{t_0} (t-t_0) \right) dt \\ &\quad + \int_{t_0+\delta t}^{t_f} G(\mathbf{x}(t),\dot{\mathbf{x}}(t),t)\,dt \end{aligned} \tag{10}$$

Now assume that $G(\cdot)$ has no dependence on $\dot{\mathbf{x}}(t)$ or t. Then,

$$\begin{aligned} J[\mathbf{x}(t),\dot{\mathbf{x}}(t),t] &= F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) \\ &\quad + \int_{t_0}^{t_0+\delta t} \left(G(\mathbf{x}(t_0)) + \left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} (\mathbf{x}(t)-\mathbf{x}(t_0)) \right) dt \\ &\quad + \int_{t_0+\delta t}^{t_f} G(\mathbf{x}(t))\,dt \\ &= F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) \\ &\quad + G(\mathbf{x}(t_0))\delta t - \left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} \mathbf{x}(t_0)\delta t + \left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} \int_{t_0}^{t_0+\delta t} \mathbf{x}(t)dt \\ &\quad + \int_{t_0+\delta t}^{t_f} G(\mathbf{x}(t))\,dt \end{aligned} \tag{11}$$

Finally, apply a trapezoidal integration scheme to approximate the value of the first integral term:

$$\begin{aligned} J[\mathbf{x}(t),\dot{\mathbf{x}}(t),t] &= G(\mathbf{x}(t_0))\delta t - \frac{\delta t}{2}\left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} \mathbf{x}(t_0) + \frac{\delta t}{2}\left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} \mathbf{x}(t_0+\delta t) \\ &\quad + \int_{t_0+\delta t}^{t_f} G(\mathbf{x}(t))\,dt + F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) \end{aligned} \tag{12}$$

Now consider the problem of finding the position of the waypoint $\mathbf{x}(t_0+\delta t)$ which minimizes Equation 12. First notice that only the last three terms in Equation 12 depend on $\mathbf{x}(t_0+\delta t)$ (the dependence of the last term is indirect), so we may without loss of generality consider minimizing

$$\frac{\delta t}{2}\left.\frac{\partial G}{\partial \mathbf{x}}\right|_{x_0} \mathbf{x}(t_0+\delta t) + \int_{t_0+\delta t}^{t_f} G(\mathbf{x}(t))\,dt + F(\mathbf{x}(t_f),\dot{\mathbf{x}}(t_f),t_f) \tag{13}$$

The first term may be interpreted as the immediate cost, since it depends only on the local gradient of the cost functional. The other two terms may be interpreted as the true cost-to-go $E[\cdot]$. If the cost-to-go terms are ignored, then it is clear that $\mathbf{x}(t_0 + \delta t)$ must lie along $-\left.\partial G/\mathbf{x}(t)\right|_{t_0}$ in order to minimize the immediate cost. This is exactly how APFG chooses the direction of the next waypoint. Thus, APFG may be interpreted as an algorithm which discounts any future cost and attempts to minimize only the cost of the next step. Alternatively, it is equally valid to say that the APFG attractive potential is a cost-to-go term,

$$E(\mathbf{x}(t+\delta t), \dot{\mathbf{x}}(t+\delta t), t+\delta t) = A(\mathbf{x}(t+\delta), \mathbf{x}_d)$$

so that APFG is in fact attempting to minimize a cost functional of the form

$$J[\mathbf{x}(t), t] = \int_t^{t+\delta t} \beta \sum_{j=1}^{n} R(\mathit{dist}(\mathbf{x}(t), c_j))\, dt + E(\mathbf{x}(t+\delta t)).$$

Thus, an APFG planner can be seen as a receding horizon planner with a vanishingly small horizon.

5 Discussion

The insight that APFG can be interpreted as a receding horizon planner with a vanishingly small horizon immediately suggests a method for adapting receding horizon planners for use in an anytime planning [3] framework. Anytime planning is a generic term describing planning algorithms that can provide a solution that satisfies a minimal set of requirements within a short, known timeframe, and then improves on that solution for however much time is available. Notably, a valid solution is always available after the minimum initial planning time. As a consequence, anytime planners are very suitable for real–time applications.

There are two challenges to applying anytime planning ideas to a receding horizon planner. First, a hard guarantee is required for the maximum amount of time before which a minimally satisfactory trajectory is available. Second, there must be a way of iteratively improving the solution while guaranteeing that a valid solution is always available. Both of these challenges can be solved for a receding horizon trajectory planner by starting with a very small planning horizon. This allows the first planning iteration to proceed using APFG, which in turn provides an absolute maximum computation time guarantee. The succeeding iterations then proceed using the standard receding horizon planner, with the planning horizon increasing at each iteration. The results from the previous iteration are always available, thus satisfying the second challenge. This scheme should scale gracefully with time: when the available computation time is short, the planner is guaranteed to provide results at least as good as an APFG planner. As available computation time increases, the results approach those provided by a true end-to-end optimal planner. This second point is an important benefit of anytime planning: instead of sitting idle after the required computations are completed, an anytime planner utilizes the remaining time to improve the quality of the solution. Thus, an anytime planner will certainly guarantee real–time execution, and may, depending on the specifics of the problem at hand, return better results than a standard receding horizon planner.

6 Preliminary Experimental Results

6.1 Planar holonomic navigation in a cluttered environment

As a first step to validating the ideas of the previous section, an experiment was designed to compare the performance of APFG and of optimal motion planning with a receding horizon planner. The problem domain entails a holonomic robot moving in a two-dimensional square room with four walls and dimensions of 1.2×1.2. Ten circular obstacles, two of diameter 0.142, five of diameter 0.1, and three of diameter 0.05, are placed randomly within the room such that there is a minimum distance of at least 0.08 between each other and from the walls. The robot is required to begin in the lower-left hand corner of the room, $\mathbf{x}_0 = (0, 0)$, and travel to the upper-right hand corner, $\mathbf{x}_f = (1, 1)$ while avoiding collisions with the walls and the circular obstacles. The initial time is $t_0 = 0$ and the final time is $t_f = 10$.

The cost functional used for the optimal trajectory planner is:

$$J[\mathbf{x}(t)] = \int_{t_0}^{t_f} \ddot{\mathbf{x}}(t)^T \ddot{\mathbf{x}}(t) + \beta \sum_{j=0}^{n} \sum_{k=0}^{m} R(\mathit{dist}(\mathbf{b}_k, c_j, t_k, t_{k+1}))\, dt \tag{14}$$

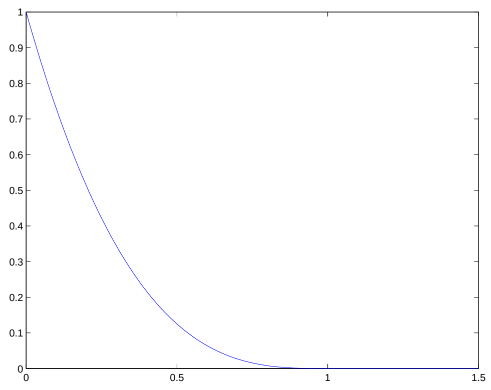

Figure 1: Piecewise cubic obstacle penalty function, $R(\cdot)$, with $k = 1$ and $d_n = 1$.

where $R(\cdot)$ is here defined as a piecewise cubic:

$$R(d) = \begin{cases} k \cdot \left(a_1(d/d_n)^3 + a_2(d/d_n)^2 + a_3(d/d_n) + a_4\right) & d < d_n \\ 0 & \text{else.} \end{cases}$$

Here, d is the distance from the robot to an obstacle, d_n is the distance over which $R(\cdot)$ has nonzero support, and k is the penalty at $d = 0$. The constants a_1, a_2, a_3, and a_4 are chosen so that $R(0) = 1$, $R(d_n) = 0$, $\dot{R}(d_n) = 0$, and $\ddot{R}(d_n) = 0$. The last two constraints are important to ensure that the smoothness requirements of the vector minimization algorithm are met. The distance function is defined as

$$\textit{dist}(\mathbf{b}, c, t_k, t_{k+1}) = \min_{t \in (t_k, t_{k+1})} \min_{y \in c} \|\mathbf{x}(t) - y\|.$$

i.e. it returns the minimum Euclidean distance between the obstacle and the path between knots t_k and t_{k+1} where $\mathbf{b}$ is the vector of weights that defines $\mathbf{x}(t)$ on the interval (t_k, t_{k+1}). With the circular objects the distance was calculated from the obstacle center. This is necessary because with nonlinear minimization problems, intermediate estimates of the optimal path are not guaranteed to be collision free, and in order for the minimization algorithm to proceed the cost functional and its gradient must be well defined in such a case. Therefore, *dist*$(\cdot)$ must be defined for points inside the boundary of an obstacle. Defining the distance as zero at the center of the obstacle is one way to accomplish this.

The support length d_n was chosen to be 0.05 plus the obstacle diameter. The penalty function height term k was chosen so that $R(\cdot) = 1$ at the obstacle boundary. The weight penalty term β was chosen as 4.5.

The direct method was used to solve this optimization problem. The true optimum path $\mathbf{x}^*(t)$ was approximated with a fourteen knot third-order Hermite spline. The knots were uniformly distributed between t_0 and t_f. The vector minimization algorithm chosen was the Broyden-Fletcher-Goldfarb-Shanno routine of the Gnu Scientific Library (http://www.gnu.org/software/gsl/). BFGS is a quasi-Newton algorithm that internally estimates the Hessian matrix. Because even the first derivative of $J[\cdot]$ is difficult to derive symbolically, it was estimated numerically.

For the APFG planner the potential function was:

$$J[\mathbf{x}(t)] = A(\mathbf{x}(t), \mathbf{x}_f) + \beta \sum_{i=0}^{n} R(\textit{dist}(\mathbf{x}(t), c_i)) \tag{15}$$

where $A(\cdot)$, the attractive potential term, is defined as:

$$\begin{aligned} d &= \|\mathbf{x} - \mathbf{x}_f\| \\ A(\mathbf{x}, \mathbf{x}_f) &= \begin{cases} d & \text{if } d > \epsilon \\ \frac{1}{2\epsilon} d^2 + \epsilon/2 & \text{else.} \end{cases} \end{aligned}$$

A graph of this function is shown if Figure 2. This attractive term is used in lieu of the more common quadratic term shown in Equation 2 in order to minimize the size of the initial velocity spike discussed in Section 2. This form also allows the

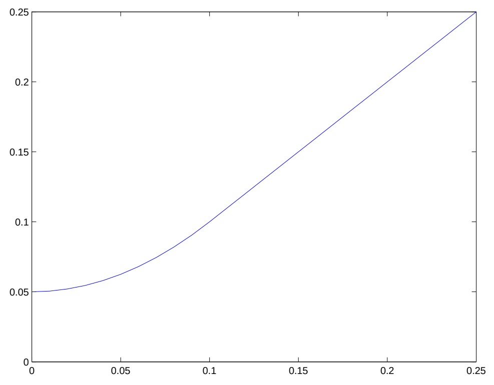

Figure 2: APFG planner attractive potential term.

planner to take steps of constant length in the absence of obstacles except in the immediate vicinity of the goal; a quadratic term leads to longer steps as the distance from the goal increases, which essentially means that the obstacle avoidance term becomes is discounted far from the goal.

The constants β, d_n, and k were set to the same values as for the optimal trajectory planner. Given waypoint $\mathbf{x}(t_i)$, the succeeding waypoint $\mathbf{x}(t_i + \delta t) = \mathbf{x}(t_{i+1})$ was chosen as

$$\mathbf{x}(t_i + \delta t) = \mathbf{x}(t_i) - \gamma \left. \frac{\partial J}{\partial \mathbf{x}(t_i)} \right|_{\mathbf{x}(t_i)}$$

At each waypoint, the velocity to be achieved was

$$\mathbf{v}_i = (\mathbf{x}_i - \mathbf{x}_{i-1})/\delta t$$

except for the initial and final states, where $\mathbf{v}_0 = 0$ and $\mathbf{v}_f = 0$. The trajectory was parameterized as a third-order Hermite spline with knots at each t_i and the cost was evaluated using Equation 14.

Finally, for the limited-horizon planner the cost functional used was

$$J[\mathbf{x}(t)] = \int_t^{t+h} \ddot{\mathbf{x}}(t)^T \ddot{\mathbf{x}}(t) + \beta \sum_{i=0}^{n} \sum_{j=0}^{m} R(\mathit{dist}(\mathbf{b}_j, c_i, t_j, t_{j+1}))\, dt + \int_{t+h}^{t_f} \ddot{\mathbf{x}}(t)^T \ddot{\mathbf{x}}(t)\, dt \tag{16}$$

where $h = 1.5$. As in the other two cases, third-order Hermite splines were used to parameterize the trajectory $\mathbf{x}(t)$; in this case, the spline has two knots, at t_i and $t_{i+1} = t + h$. Since the positions $\mathbf{x}(t_i)$ and $\mathbf{x}(t_f)$ are fixed, the only degrees of freedom for the optimization routine to vary are the weights associated with the second knot. Following each iteration, a step of 1 time unit was taken along the spline, and the new optimization was started at this point.

Ten different configurations of obstacles were randomly generated; a sample configuration is shown in Figure 3. The resulting trajectories were all evaluated according to equation 14. Data was collected on a 1.25 GHz G4 computer. The trajectory costs and computational times for the three trajectory planners are shown in Figure 4.

Notice that the receding horizon planner does in fact represent an intermediate step between APFG and optimal motion planning. Its execution time compares favorably with the APFG planner, while the quality of the trajectories it finds are close to those of the optimal trajectory planner.

Notice that in one case, Example 6, the cost of the trajectory found by the receding horizon planner is actually lower than that of the optimal motion planner. This is an example of the inability of nonlinear optimization routines to converge to a global minimum discussed in the Section 3. Examination of the two trajectories shows that the optimal motion planner takes a significantly different route through the obstacle field than the receding horizon planner does; in this case, the receding horizon planner happens to find a trajectory that is much closer to the global minimum.

It is also worth mentioning that the data illustrates one of the limitations of nonlinear optimization routines discussed in Section 3. Occasionally, the receding horizon trajectory planner encountered a case where the gradient norm convergence criterion failed to accurately detect convergence. The iteration terminated when it was unable to improve on the trajectory cost. As mentioned, this causes the iteration to take longer than is strictly necessary, but does not indicate a failure to

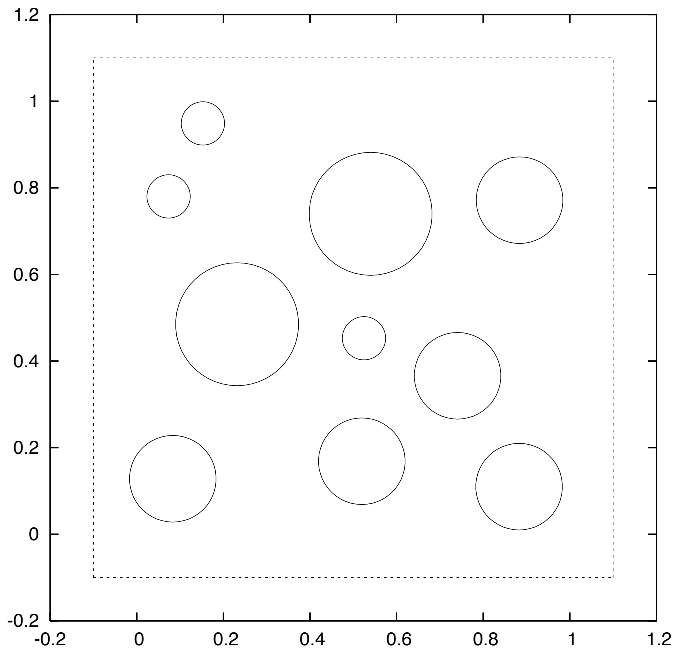

Figure 3: Typical experimental configuration.

	Optimal		APFG		Receding Horizon	
	Cost	Time	Cost	Time	Cost	Time
Example 1	0.150	91.41	5.579	0.24	0.539	2.48
Example 2	0.203	196.94	4.667	0.29	0.330	1.88
Example 3	0.066	64.82	4.896	0.19	0.180	1.39
Example 4	0.060	77.11	5.352	0.25	0.284	1.98
Example 5	0.158	171.09	5.198	0.32	0.281	2.98
Example 6	0.161	230.65	5.166	0.27	0.075	0.41
Example 7	0.153	246.18	5.004	0.21	0.174	0.63
Example 8	0.134	253.61	5.314	0.31	0.241	0.94
Example 9	0.342	391.4	4.548	0.24	0.156	0.91
Example 10	0.076	194.38	4.063	0.29	0.214	1.72
Average	0.1503 ± 0.0822	191.759 ± 98.852	4.9787 ± 0.4502	0.261 ± 0.043	0.2474 ± 0.1263	1.532 ± 0.829

Figure 4: Experimental results comparing the performance of the optimal, APFG, and receding horizon planners. All times are in seconds.

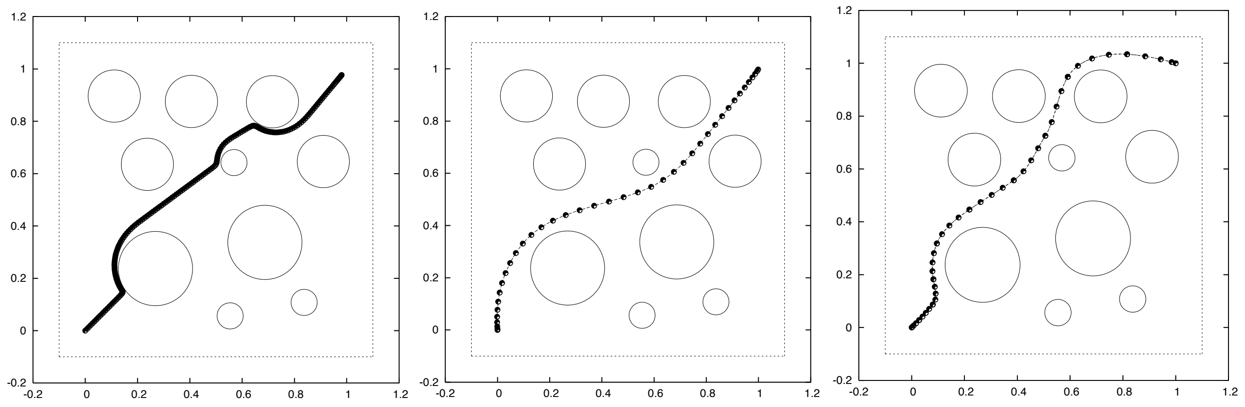

Figure 5: Results for Example #4; from left to right, APFG, optimal motion planner, and receding horizon planner.

converge; the trajectory returned by the planner is still useable. Even in the face of such problems, however, the receding horizon planner still runs relatively quickly.

6.2 Orbital robotic grapple

Next, the receding horizon planning was applied to a more realistic problem: grappling a disabled satellite using a robotic manipulator. Robotic arm trajectory planning is extremely difficult for several reasons: robot arms possess both nonlinear kinematics and dynamics, including significant singularities and joint limitations; robot arms have a complex shape, and are often used in cluttered environments near objects with complex shapes, making distance calculations expensive; and robot arms have many degrees of freedom, so the trajectory planner must operate in a space of six or more dimensions.

The results presented here entail grappling a docking point located on the aft end of a generic satellite, shown in Figure 6. Notice that the satellite has a sun shade surrounding the workspace, and that the grapple fixture is located approximately 6 cm. from the shade. In addition, a rocket nozzle and various other pieces of equipment are present. The robot arm, shown in Figure 7 is modelled on a Mitsubishi Heavy Industries PA-10/7C, a common 7 degree–of–freedom industrial manipulator. The cost functional used here was:

$$J[\theta(t)] = \int_t^{t+h} \ddot{\theta}(t)^T\ddot{\theta}(t) + \beta\sum_{i=0}^{n} R(\mathit{dist}(\theta(t+h), c_i))\; dt + \int_{t+h}^{t_f} \ddot{\theta}(t)^T\ddot{\theta}(t)\; dt + \|\mathbf{x}(\theta(t_f)) - \mathbf{x}_d\| + \sum_{i=0}^{n} R(\mathit{dist}(\theta(t_f), c_i)) \tag{17}$$

where, here, $\theta(t)$ is a vector of joint angles, $\mathbf{x}(\theta)$ represents the cartesian coordinates of the end effector given joint angles θ, and $\mathbf{x}_d$ represents the desired final end effector cartesian position and pose. Notice that two additional terms have been added to the cost–to–go estimate: a term that penalizes deviations from the desired final end effector position and pose, and a term that penalizes obstacle distance at the final position. For this problem, the final arm position $\theta(t_f)$ was not fixed; instead, these two terms were used allow the trajectory planner to use the extra degree of freedom of the robot arm to fulfill the required final conditions while maximizing obstacle distances. For this application the arrival time t_f was 30 seconds; the horizon h was 6 seconds and the step size δ was 3 seconds.

Distance calculation for this example is considerably more complicated and computationally expensive than the previous example. To reduce the computation time, instead of attempting to find the minimum distance between the robot and the obstacles anywhere in the interval $(t, t+h)$ as in the previous example, here distance was calculated only at $t+h$. Thus, in a slight abuse of notation, the distance function $R(\theta(t), c)$ here is defined as the instantaneous distance between the robot arm with joint angles θ and obstacle c.

Ten total trials with varying relative cartesian positions were conducted. The relative positions, which were measured from the base of the robot arm to the base of the rocket nozzle on the target, were in the range $x \in [0.915, 1.215]$, $y \in [0.23, 0.29]$, $z \in [0.23, 0.29]$ meters. Data was collected on a 1.8 GHz dual processor G5 computer. An example trajectory with a relative pose of $[0.965, 0.27, 0.27]$ meters is shown in Figure 8. The average time to calculate a step was 0.88 ± 0.57 seconds. The maximum time was 3.78 seconds. Note that, because on average the step sizes are larger than the calculation time required to compute them, the time step the planner is calculating may be significantly ahead of the

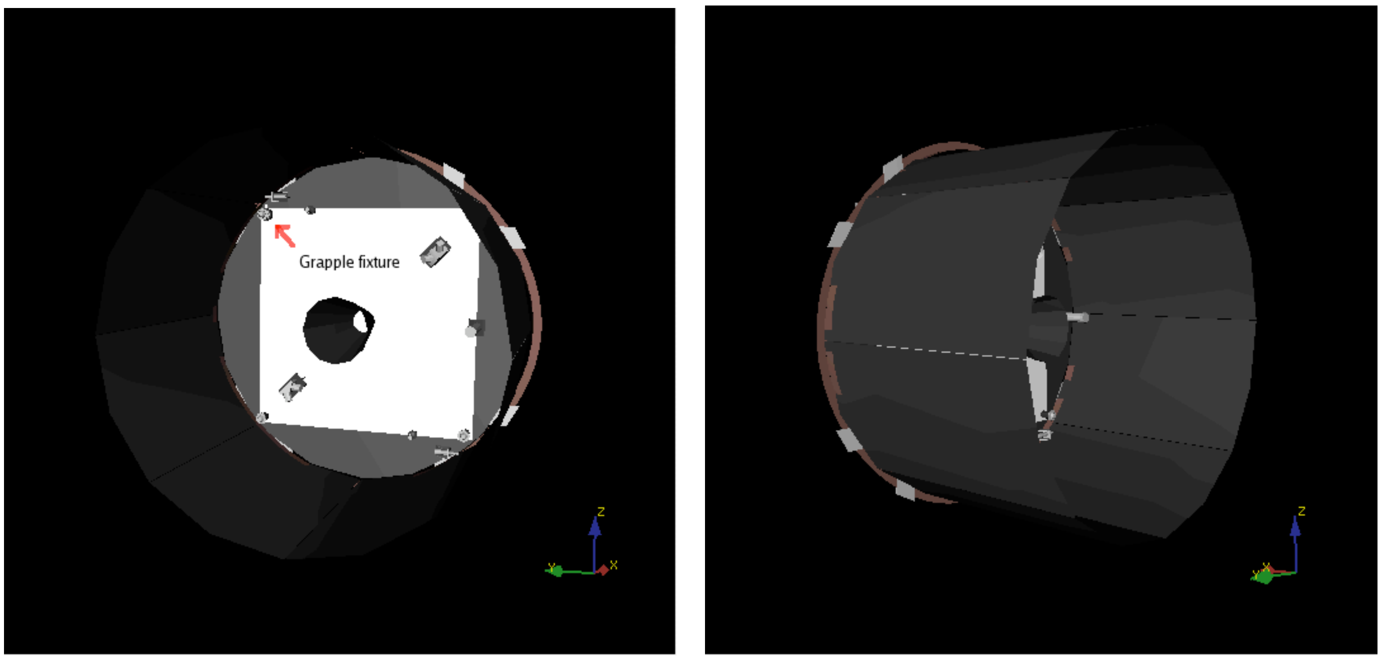

Figure 6: Satellite target for robotic arm grapple. Grapple fixture is in upper left corner of target. Target is approximately 1.83 meters (6 feet) in diameter; sunshade is approximately 1.22 meters (4 feet) deep.

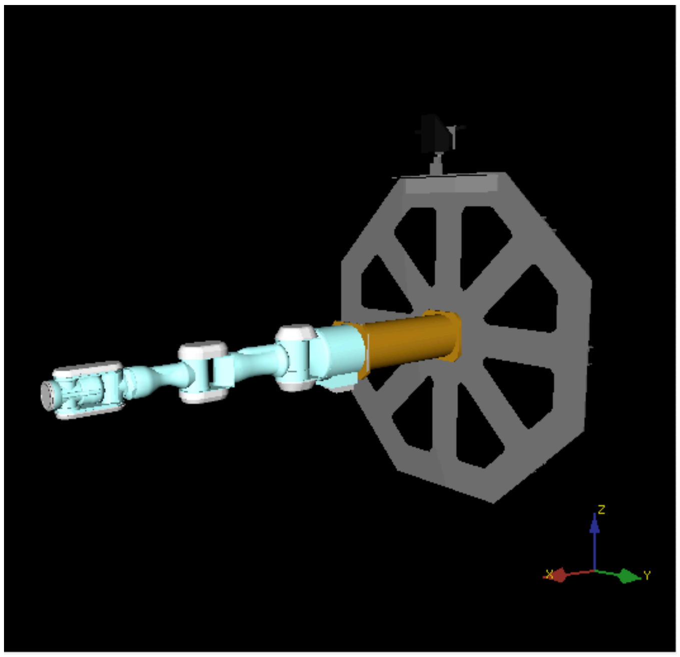

Figure 7: Robotic satellite servicer

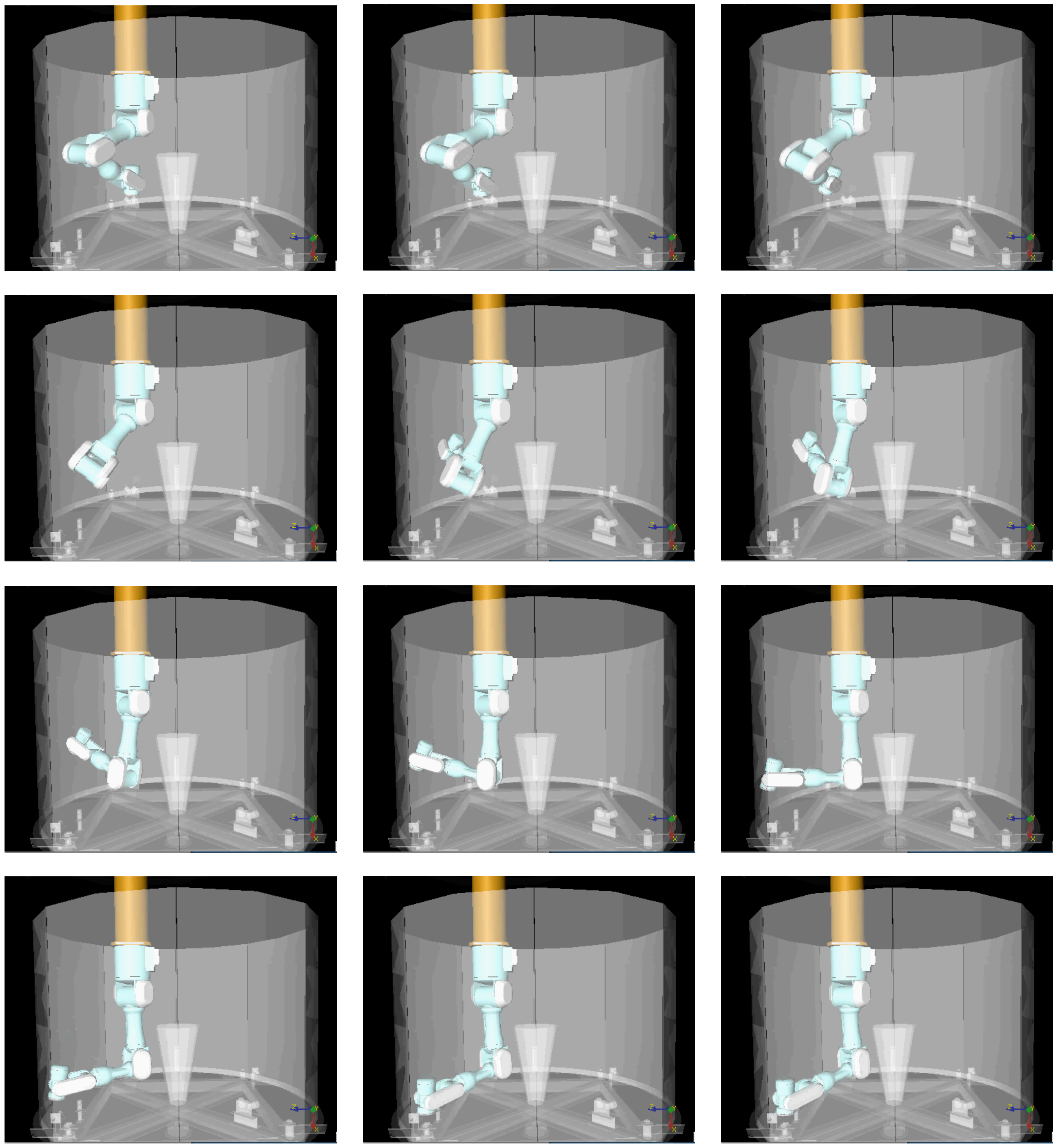

Figure 8: Successful robotic arm grapple. Images shown at approximately 2.75 second intervals.

actual elapsed time. Thus, having an individual step that takes longer than the step lengthδ to calculate does not necessarily indicate that the planner has exceeded its allotted calculation time. However, if the planner is not far enough ahead of the actual elapsed time, it *may* exceed the allotted time. Thus, while the receding horizon planner does on average satisfy the real time requirements of this particular task, the fact that it took longer than the allocated time to calculate for one iteration reinforces the need for the anytime planner discussed in Section 5.

7 Conclusions

The results shown here indicate that receding horizon trajectory planning represents an improvement over both optimal motion planning and APFG in that it allows the use of cost functional optimization, with its attendant flexibility, in a real–time setting. The first experimental setup is very similar to one of the most common robot motion planning problems, that of finding trajectories through a room or hallway, and the results show that receding horizon motion planning is capable of solving this problem quickly and of returning trajectories of much higher quality than APFG. The second experimental setup is an extremely challenging, realistic problem of current interest; the receding horizon planner was shown to be capably of ably solving this problem in real time using a current desktop–class computer.

There are several areas of future work which require investigation. First, as discussed earlier, nonlinear receding horizon planners do not satisfy hard real–time constraints. Implementing the anytime planner discussed in Section 5 constitutes the bulk of our current work. This is necessary to formally fulfill hard real–time constraints, and as was indicated by the maximum computation time in the second example, this problem is not merely a formal nicety.

There are several related topics which require more investigation. As mentioned in Section 2, APFG planners are susceptible to being caught in local minima of the potential field and never reaching the goal state. This is a result of their local nature. Receding horizon planners are not entirely local – they utilize information about the cost functional at some distance from the current step into account – but they are not entirely global either. As a result, they may be susceptible to the same problem, although probably not to the same degree. Work on characterizing the receding horizon planner's behavior regarding this problem is ongoing.

Reducing the difficulty of designing cost functionals is also an area deserving consideration. It is not always easy to design cost functional that results in reasonable performance. One possible approach to alleviating this problem might be to use on offline optimization routine such as genetic algorithms to automate the cost functional design process. A related problem is that of designing a cost–to–go estimate that is computationally inexpensive to compute but leads to optimal or nearly optimal trajectories. It may also be possible to design a cost–to–go estimate that alleviates the local minima problem discussed above.

Motion planning in the face of complex dynamics and kinematics, cluttered environments and moving obstacles and targets remains a very difficult problem. It will likely never be fully solved in its most general form. Hopefully, the technique presented here constitutes a step forward in motion planning capabilities, and will enable solutions to problems of current interest that are not current tractable.

References

[1] John Bellingham, Arthur Richards, and Jonathan P. How. Receding horizon control of autonomous aerial vehicles. In *Proceedings of the 2002 American Control Conference*, Anchorage, AK, USA, May 2002.

[2] Christopher I. Connelly. Harmonic functions and collision probabilities. In *Proceedings of the 1994 IEEE Conference on Robotics and Auomation*, San Diego, CA, 1994.

[3] T. L. Dean and M. Boddy. An analysis of time–dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Minneapolis, Minnesota, 1988.

[4] Rolf Findeisen and Frank Allgower. n introduction to nonlinear model predictive control. In *21st Benelux Meeting on Systems and Control*, Veldhoven, 2002.

[5] Carl Glen Henshaw. *A Variational Technique for Spacecraft Trajectory Planning*. PhD thesis, University of Maryland, College Park, Maryland, 2003.

[6] O. Khatib. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, 1980.

[7] O. Khatib. Real time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1), Spring 1986.

[8] Jin-Oh Kim and Pradeep K. Khosla. Real–time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation*, 8(3), June 1992.

[9] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Prentice–Hall, 1970.

[10] R. Lampariello, S. Agrawal, and G Hirzinger. Optimal motion planning for free-flying robots. In *Proceedings of the 2003 International Conference on Robotics and Automation*, Taiwan, 2003.

[11] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[12] Colin R. McInnes. Autonomous path planning for on–orbit servicing vehicles. In *Reducing Space Mission Cost*. British Interplanetary Society, April 1999.

[13] F. McQuade and C. R. McInnes. Autonomous control for on–orbit assembly using potential function methods. *The Aeronautical Journal*, pages 255–262, June/July 1997.

[14] Mark B. Milam, Kudah Mushambi, and Richard M. Murray. A new computational approach to real–time trajectory generation for constrained mechanical systems. In *Proceedings of the 2000 IEEE Conference on Decision and Control*, 2000.

[15] James B. Rawlings. Tutorial overview of model predictive control. *IEEE Control Systems Magazine*, pages 38–52, June 2000.

[16] Arthur Richards, Tom Schouwenaars, Jonathan P. How, and Eric Feron. Spacecraft trajectory planning with collision and plume avoidance using mixed–integer linear programming. *Accepted, AIAA Journal of Guidance, Control and Dynamics*, July 2002.

[17] Elan Rimon and Daniel E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5), October 1992.

[18] Alexander B. Roger and Colin R. McInnes. Safety constrained free–flyer path planning at the international space station. *Journal of Guidance, Control, and Dynamics*, 23(6):971–979, November–December 2000.

[19] T. Schouwenaars, E. Feron, and J. How. Safe receeding horizon path planning for autonomous vehicles. In *40th Allerton Conference on Communication, Control and Computing*, October 2002.

[20] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *Proceedings of the 2004 American Control Conference*, Boston, MA, USA, July 2004.

[21] Ronald J. Simmons, Edward V. Bergmann, Bruce A. Persson, and Walter M. Hollister. Six dimensional trajectory solver for autonomous proximity operations. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, pages 1291–1303. American Institute of Aeronautics and Astronautics, 1990. AIAA paper 90–3459.

[22] Kimon P. Valavanis, Timothy Hebert, Ramesh Kolluru, and Nikos Tsourveloudis. Mobile robot navigation in 2-d dynamic environments using an electrostatic potential field. *IEEE Transactions on Systems, Man, and Cybernetics –Part A: Systems and Humans*, 30(2), March 2000.

[23] Michiel J. Van Nieuwstadt and Richard M. Murray. Real time trajectory generation for differentially flat systems. Division of Engineering and Applied Science, California Institute of Technology. Submitted, International Journal of Robust and Nonlinear Control, May 1997.

[24] Oscar von Stryk. Numerical solution of optimal control problems by direct collocation. In R. Bulirsch, A. Miele, J. Stoer, and K.-H. Well, editors, *Optimal control — Calculus of Variations, Optimal Control Theory and Numerical Methods*, number 111 in International Series of Numerical Mathematics. Birkhauser, Basel, 1993.